CptS 515 Advanced Algorithms HOMEWORK-2

Sharath Kumar Karnati

October 2023

QUESTION 1:

In Lesson 3, we talked about the Tarjan algorithm (SCC algorithm). Now, you are required to find an efficient algorithm to solve the following problem. Let G be a directed graph where every node is labeled with a color. Many nodes can share the same color. Let v_1, v_2, v_3 be three distinct nodes of the graph (while the graph may have many other nodes besides the three). I want to know whether the following items are all true: there is a walk α. from v_1 to v_2 and a walk β from v_1 to v_3 such that

- α is longer than β;
- α contains only red nodes (excluding the two end nodes);
- β contains only green nodes (excluding the two end nodes).

Solution:

Now, Firstly, lets check that whether there exists any walk (β) between v_1 and v_3 that only has green nodes(excluding the two end nodes). Then, we will construct a induced sub graph called as $G_(green)$, by excluding all the vertices other than v_1 , v_3 and green nodes. Now, by checking the shortest path algorithm from v_1 and v_3(similar to Dijkstra), we will get these following conditions.

1.In $G_(green)$, there will be no path from v_1 and v_3 .This implies that a walk β does not exists

2.In $G_(green)$, there will be a finite path that connects v_1 and v_3 .So, every path from v_1 and v_3 will be a walk. Hence, we have to find a longer path β that will only contain green nodes as the condition given implies.

Then, lets check whether there exists any walk(α) between v_1 and v_2 that only contains red nodes(excluding the two end nodes) and also should be longer than β that we found from above. Now, we will construct a induced sub graph called as $G_(red)$, by excluding all the vertices other than v_1 , v_2 and red coloured nodes. The following conditions we get from this are:

1.In $G_(red)$, there will be no path to go from v_1 and v_2. This implies that walk α does not exists and the whole condition will become false.

2.If v_1 and v_2 exists in a same SCC in $G_(red)$ or the path between v_1 and v_2 has an SCC of size 2 or more, then there will be an infinite walk α from v_1 and v_2.

3.If not, we will check $G_(red)$ for all the possible paths between v_1 and v_2 and verify where it is longer than the walk β that is between v_1 and v_3.

ALGORITHM: To check whether α path is longer than β path .

INPUT : v_1, v_2 and v_3 nodes and two induced graphs.
RESULT: Boolean Values(True/False) according to the conditions.
Step 1: // Create $G_(green)$ by only considering green nodes, v_1 and v_3. and neglect all other nodes.
Step 2: // Use Dijkstra algorithm on $G_(green)$.
Step 3: If there is no path between v_1 and v_3:
Step 4: Return False
Step 5: else
Step 6: $\beta \leftarrow$ Dijkstra($G_(green)$,v_1 and v_3).
Step 7: end if
Step 8: // Create sub-graph $G_(red)$ by only considering red nodes, v_1 and v_2 and neglect all other nodes.
Step 9: // Use Dijkstra algorithm on $G_(red)$.
Step 10: if there is no path between v_1 and v_2:
Step 11: Return False
Step 12: end if
Step 13: // Use Tarjan SCC algorithm on all $G_(red)$ components to compute them.
Step 14: if v_1 and v_2 are in same SCC then
Step 15: Return True
Step 16: end if
Step 17: // Run DFS starting from v_1 on $G_(red)$ where each SCC is treated as super node if the size of any SCC is greater than 1.
Step 18: Function longestα(i,j):
Step 19: if (i==j):
Step 20: return 0
Step 21: else
Step 22: k(temporary)$\leftarrow$ 0(empty) // empty list to store the length of longest path of i neighbours
Step 23: for each neighbour x of i :
Step 24: if x is a SCC with 2 or more nodes:
Step 25: return infinity(∞)
Step 26: else append longestα(i,j)+1 (add to list k)
Step 27: return max(k)
Step 28: end Function
Step 29: if longestα(v_1,v_2) $>$ β then;
Step 30: return True
Step 31: else
Step 32: return False
Step 33: end if.

QUESTION 2: In Lesson 4, we learned network flow. In the problem, capacities on a graph are given constants (which are the algorithm's input, along with the graph itself). Now, suppose that we are interested in two edges e1 and e2 whose capacities c1 and c2 are not given but we only know these two variables are non negative and satisfying c1+c2 ¡ K where K is a given positive number (so the K is part of the algorithm's input). Under this setting, can you think of an effcicient algorithm to solve network flow problem? This is a difficult problem.

SOLUTION:

We will slove this problem by reduction into a Linear Programming Formulation.Lets assume that, in G we will have a source(s) and a sink(t). Let assume f(p,q) is a flow variable to each edge (p,q) in a edge set E of G.
Then,
The Maximum of the flow can be respresented as :
$MaxFlow = \sum_{q(s,p)\in E}\ f(s,\ p)$
Now,
The equation for the flow can be given as:
$\sum_{p:(p,q)} f(p,q) = \sum_{w:(q,w)} f(q,w), \forall q \in V-(s,t)$

The capacity bounds and non negative flow will be represented as:

$f(p,q) \leq c(p,q), \forall p,q \in E$ and $f(p,q) \geq 0, \forall p,q \in E$

Let us assume that two edges e1 and e2 with variable capacities c1 and c2 be given as $(u,v) \in Eand(r,s) \in E$.

Now both are variable capacities so by adding them,

The final constraint equation will be:
$c1 + c2 < K = f(u,v) + f(r,s) < K$

As all the equations and constraints are linear we can easily solve this problem by using any linear program,ming algorithms like Simplex.

QUESTION 3:

There are a lot of interesting problems concerning graph traversal — noticing that a program in an abstract form can be understood as a directed graph. Let G be a SCC, where v0 is a designated initial node. In particular, each node in G is labeled with a color. I have the following property that I would like to know whether the graph satisfies:

For each inifintely long path starting from v0, passes a red node from which, there is an infinitely long path that passes a green node and after this green node, does not pass a yellow node. 1

Please design an algorithm to check whether G satisfies the property

SOLUTION: We can find the solution for this problem using both Tarjan's SCC algorithm and DFS(depth first search).
The algorithm for this problem will be :

- 1.Firstly, run DFS starting from v_0. We will create a new sets called RED and GREEN to represent all red and all green nodes along the path starting from v_0.
- 2.Now, lets create a new graph called 'G_n' from the old one(G) by neglecting all the yellow nodes.
- 3.Now, lets find if there will be a green node in the set of GREEN whose SCC consists of a node size of 2 or more . We can do this by running Trajan's SCC algorithm on GREEN and if we find any of green nodes of

nodesize >= 2 in set GREEN we will put them in a new set termed as $GREEN_n$.

- 4.Now, FOR every pair of a red(i) and a green node (j) from RED and $GREEN_n$, respectively.
- 5.If there exists a path that connects i and j , RETURN TRUE.
- 6.Else, RETURN FALSE(if there will be no path that connects them).

QUESTION 4:

Path counting forms a class of graph problems. Let G be a DAG where vand v_0 be two designated nodes. Again, each node is labeled with a color.
(1). Design an algorithm to obtain the number of paths from v to v_0 in G.
(2). A good path is one where the number of green nodes is greater than the number of yellow nodes. Design an algorithm to obtain the number of good paths from v to v_0 in G.

SOLUTION (4.1):

We can use dynamic programming algorithm for this problem to count the number of paths from v to v_0 in G.
ALGORITHM: Finding number of paths from v to v_0.
Input: nodes v and v_0, A DAG G.
Result: Number of paths.
1 . // create an array whose size equal to the number of vertices which is set to zero
2. numberOfpaths[1: number of vertices] $\leftarrow$ 0
3. Function dFSCount(a,b):
4. If (a==b):
5. Return 1
6. Else:
7. numberOfpaths[a]= $Ac : (a, c) \in G$ dFSCount(c,b)
8. Return numberOfpaths[p]
9. End Function
10. Return dFSCount(v , v_0)

SOLUTION (4.2) :

We can use dynamic programming to count the number of good paths from v to v_0 in G.
ALGORITHM: To find the number of good paths from v to v_0
INPUT : A DAG G, two nodes v and v_0.
OUTPUT: Number of good paths from v to v_0.
1. // Create a array whose size is equal to the number of vertices which is intialized as zero.
2. numPaths[1: number of vertices] = empty list[]
3. Function dFSGoodpaths(i,j):
4. If (i==j) and j is a green node:
5. Return (1,0)

6. If (i==j) and j is yellow node:
7. Return (0,1)
8. Else:
9. For each x(neighbour) of i
10. numPaths[x] = dFSGoodpaths(x,j)
11. For each pair(x n-green, x n-yellow) in numPaths[x]:
12. Append (x n-green + 1 , x n-yellow) to numPaths[i] if i is a green node:
13. Else
14. Append(x n-green, x n-yellow+1) to numPaths[i] if i is a yellow node.
15. Return numPaths[i]
16. All paths ← dFSCount(v,v_0)
17. End Function
18. Returns number of good paths in all paths where n-green > n-yellow.